Nama : Tora Margaretha Chrisdya Wardani
Kelas : 2IA11
NPM : 51422591

Rangkuman Materi PBO Pertemuan 1-6

- **Pengenalan Pemrograman Berbasis Objek**
  - Pemrograman berorientasi objek (Object Oriented Programming atau disingkat OOP) adalah paradigma pemrograman yang berorientasikan kepada objek yang merupakan suatu metode dalam pembuatan program, dengan tujuan untuk menyelesaikan kompleksnya berbagai masalah program yang terus meningkat.
  - Pemrograman berorientasi objek ditemukan pada Tahun 1960, dimana berawal dari suatu pembuatan program yang terstruktur (structured programming). Metode ini dikembangkan dari bahsa C dan Pascal.
  - Pemrograman berorientasi objek dalam melakukan pemecahan suatu masalah tidak melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut
  - Pemrograman berorientasi objek bekerja dengan baik ketika dibarengi dengan Objek-Oriented Analysis And Design Process (OOAD).

PEMROGRAMAN BERBASIS OBJEK VS PEMROGRAMAN TERSTRUKTUR

- **Pemrograman berbasis objek**
  - Menggabungkan fungsi dan data dalam kelas – kelas atau objek – objek.
  - Memiliki ciri Encapsulation (pengemasan), Inheritance (penurunan sifat) dan Polymorphism (perbedaan bentuk dan perilaku).
  - Struktur program ringkas, cukup dengan membuat Objek dan class lalu bekerja berdasarkan object dan class tersebut.
  - Kode program sangat re-usable. object dan class dapat digunakan berkali-kali, sehingga dapat menghemat space memori.
  - Efektif digunakan untuk menyelesaikan masalah besar, karena OOP terdiri dari class-class yang memisahkan setiap kode program menjadi kelompok - kelompok kecil, sesuai dengan fungsinya.
  - Sulit diawal (karena harus membuat class) namun selanjutnya akan terasa mudah dan cepat.
  - Eksekusi lebih cepat karena dieksekusi bersamaan, program hanya mengatur Objek, properties dan method-nya saja.

- **Pemrograman terstruktur**
  - Memecah program dalam fungsi dan data.
  - Memiliki ciri Sequence (berurutan), Selection (pemilihan) dan Repetition (perulangan).
  - Struktur program rumit karena berupa urutan proses dan fungsi-fungsi.
  - Re-use kode program kurang.
  - Efektif digunakan untuk menyelesaikan masalah kecil dan tidak cocok untuk menyelesaikkan masalah yang rumit, karena nantinya akan kesulitan menemukan solusi permasalahan ketika terjadi error.
  - Mudah diawal, namun kompleks diproses selanjutnya.
  - Eksekusi lebih lambat karena setiap perintah dikerjakan berurutan.

- **Class pada PBO**

Class adalah cetak biru atau blueprint dari object. Class digunakan hanya untuk membuat kerangka dasar. Sebagai analogi, class bisa diibaratkan dengan kendaraan. Kita tahu bahwa kendaraan memiliki ciri-ciri seperti merk, memiliki mesin, memiliki ban, dan beberapa ciri khas lain yang menyatakan sebuah benda tersebut adalah kendaraan. Selain memiliki ciri-ciri, sebuah kendaraan juga bisa dikenakan tindakan, seperti: menghidupkan mesin atau mematikan mesin.
Contoh Program :

```
// Deklarasi Class
public class Kendaraan{
    // Ini Adalah Class, semua konstruktor, variabel, method berada disini
}
```

- **Objek dalam pemrograman**
    - Suatu Objek adalah unik, objek dapat mewakili sesuatu di dunia nyata, contohnya: orang, mobil, rekening bank dan lain sebagainya
    - Objek mirip dengan suatu rekaman (record) dalam suatu sistem berkas (misalnya rekaman karyawan).
    - Suatu objek didefinisikan berdasarkan namanya atau menggunkan kata benda, objek juga memiliki atribut dan metode.
    - Objek adalah entitas yang memiliki atribut, karakter dan kadang kala disertai kondisi. Objek mempresentasikan sesuai kenyataan seperti siswa, mempresentasikan dalam bentuk konsep seperti merek dagang, juga bisa menyatakan visualilasi seperti bentuk huruf (font).
    - Dalam pengembangan perangkat lunak berorientasi objek, objek dalam perangkat lunak akan menyimpan state-nya dalam variable dan menyimpan informasi tingkah laku (behaviour) dalam method atau fungsi-fungsi/prosedur.

Pada dasarnya ada dua karakteristik yang utama pada sebuah objek , yaitu :

1. Setiap objek memiliki atribut sebagai status yang kemudian akan disebut sebagai state.
2. Setiap objek memiliki tingkah laku yang kemudian akan disebut sebagai behaviour.
   Contoh: objek sepeda. Sepeda memiliki atribut (*state*): pedal, roda, jeruji, dan warna. Sepeda memiliki tingkah laku (*behaviour*): kecepatannya menaik, kecepatannya menurun dan perpindahan gigi sepeda.

Untuk membuat objek, bisa menggunakan kata kunci *new* yang digunakan untuk membuat objek baru, selanjutnya menentukan 3 langkah untuk membuat sebuah objek. Yaitu: mendeklarasikan variable, membuat objek baru (Instansiasi) dan pemanggilan konstruktor.
Contoh Program:

```
public class Kendaraan{

    // Konstruktor Dengan Parameter
    public Kendaraan(String nama){
        System.out.println("Nama Kendaraannya Adalah "+ nama);
    }

    public static void main(String[]args){

        // Perintah untuk membuat objek jenis
        Kendaraan jenis = new Kendaraan("Pesawat Terbang");

    }

}

// Output = Nama Kendaraannya Adalah Pesawat Terbang
```

- **Pengertian Dasar Inheritance**
    - Pewarisan merupakan sebuah bentuk “penggunaan kembali” (reusability); dimana *class* baru dibuat dari *class* yang pernah ada yang (biasanya) ditambah fasilitasnya.
    - Setiap *class* turunan dapat menjadi *class* pokok (induk) untuk *class* turunan yang akan datang.
    - Dalam pewarisan, constructor tidak diwariskan pada classturunannya, kecuali jika digunakan perintah super.
    - Pewarisan tunggal (single inheritance) merupakan pewarisan dari satu *class* pokok (induk).
    - Pewarisan ganda (multiple inheritance) merupakan pewarisan dari dua atau lebih *class* pokok.
    - Java tidak mendukung multiple inheritance

Class yang mewariskan disebut dengan *superclass / parent class / base class*, sedangkan class yang mewarisi (*class* yang baru) disebut dengan *subclass / child class / derived class.*

Untuk menerapkan inheritance, gunakan statement “*extends*”.

Keyword “*super*” digunakan oleh subclass untukmemanggil constructor, atribut dan method yang adapada *superclass*-nya.

Contoh untuk memanggil constructor milik *superclass*-nya :

```
super()
super(parameter)
```

Contoh untuk memanggil atribut dan method milik *superclass*-nya :

```
super.namaAtribut
super.namaMethod(parameter)
```

- **Pengertian Dasar Encapsulation**
  Enkapsulasi (encapsulation) merupakan cara untuk melindungi property (atribut) / method tertentu dari sebuah kelas agar tidak sembarangandiakses dan dimodifikasi oleh suatu bagian program.

- **Accessors**
  Cara untuk melindungi data yaitu dengan menggunakan access modifiers (hak akses). Ada 4 hak akses yang tersedia, yaitu default, public, protected, private

No	Modifier	Pada class dan interface	Pada method dan variabel
1	Default (tidak ada modifier)	Dapat diakses oleh yang sepaket	Diwarisi oleh subkelas dipaket yang sama, dapat diakses oleh method-method yang sepaket
2	Public	Dapat diakses dimanapun	Diwarisi oleh subkelasnya, dapat diakses dimanapun
3	Protected	Tidak bisa diterapkan	Diwarisi oleh subkelasnya, dapat diakses oleh method-method yang sepaket
4	private	Tidak bisa diterapkan	Tidak dapat dikases dimanapun kecuali oleh method-method yang ada dalam kelas itu sendiri

- **Polymorphism**

  - Overloading
    Overloading adalah diperbolehkannya dalam sebuah class memiliki lebih dari satu nama function/method yang sama tetapi memiliki parameter/argument yang berbeda.

```
1  public class Overloading {
2      public void Tampil(){
3          System.out.println("I love Java");
4      }
5      public void Tampil(int i){
6          System.out.println("Method dengan 1 parameter = "+i);
7      }
8      public void Tampil(int i, int j){
9          System.out.println("Method dengan 2 parameter = "+i+" & "+j);
10     }
11     public void Tampil(String str){
12         System.out.println(str);
13     }
14
15     public static void main(String a[]){
16         Overloading objek = new Overloading();
17         objek.Tampil();
18         objek.Tampil(8);
19         objek.Tampil(6,7);
20         objek.Tampil("Hello world");
21     }
22 }
```

    Overriding method adalah kemampuan dari *subclass* untuk memodifikasi method dari *superclass*-nya, yaitu dengan cara menumpuk (mendefinisikan kembali) method *superclass*-nya. Contoh overriding method dapat dilihat pada subclass “Mobil” yang mendefinisikan kembali method keterangan() dan hapus() dari *class* “Kendaraan”.

- **Inheritance**

Untuk setiap burung, ada satu set properti yang telah ditetapkan yang umum untuk semua burung dan ada satu set properti yang khusus untuk burung tertentu. Oleh karena itu, secara intuitif, kita dapat mengatakan bahwa semua burung mewarisi ciri-ciri umum seperti sayap, kaki, mata, dll. Oleh karena itu, dalam cara berorientasi objek untuk merepresentasikan

burung, pertama-tama kita mendeklarasikan kelas “Bird” dengan seperangkat properti yang umum untuk semua burung. Dengan melakukan ini, kita dapat menghindari menyatakan sifat-sifat umum ini di setiap burung yang kita buat. Sebagai gantinya, kita cukup mewariskan kelas “Bird” di semua.

```
// Implementing the bird class
public class Bird {

    // Few properties which
    // define the bird
    String animal = "All Birds";
    String color;
    int legs;

    // Few operations which the
    // bird performs
    public void eat()
    {
        System.out.println(
            "This bird has eaten");
    }

    public void fly()
    {
        System.out.println(
            "This bird is flying");
    }
}
```

Setelah kelas “Bird” dibuat, jika kita ingin membuat kelas “Pigeon”, maka kita cukup mewarisi kelas “Bird” di atas.

```
// Creating the Pigeon class which
// extends the bird class
public class Pigeon extends Bird {

    String animal ="Pigeon";
    // Overriding the fly method
    // which makes this pigeon fly
    @Override
    public void fly() //override method
    {
        System.out.println(
            "Pigeon flys!!!");
    }
}
```

Dan jika kita ingin membuat method yang menggunakan atribut dari kelas ”Bird”, maka kita dapat memanggilnya dengan menggunakan keyword “super”

```
// Creating the Pigeon class which
// extends the bird class
public class Pigeon extends Bird {

    String animal ="Pigeon";
    // Overriding the fly method
    // which makes this pigeon fly
    @Override
    public void fly() //override method
    {
        System.out.println(
            "Pigeon flys!!!");
    }

    //super & this
    public void about(){
        System.out.println(super.animal+" live freely in the sky");
        System.out.println(this.animal+ " can recognise each letter of the human alphabet");
    }
}
```

- **Encapsulation**

Sekarang, kita telah mendefinisikan properti dari kelas “Bird” dan atribut yang dimiliki burung seperti warna, sayap, kaki dapat diinisialisasi dengan membuat objek kelas burung. Namun, jika kita hanya dapat mengubah properti dari kelas “Bird” hanya dengan referensi dari objek, maka atribut kehilangan informasi yang awalnya diinisialisasi.

Misalnya, katakanlah kita awalnya membuat merpati dengan warna abu-abu dengan membuat konstruktor, setiap pengguna dengan instance objek merpati dapat mengubah warna ini menjadi merah atau hitam hanya dengan merujuk atribut dengan kata kunci “this". Untuk menghindari hal ini, kita menyertakan properti ke dalam sebuah metode. Metode ini disebut getter dan setter atribut. Idenya adalah untuk hanya menyertakan inisialisasi dan pengambilan atribut dalam suatu metode alih-alih langsung merujuk atribut secara langsung. Ini juga memberikan keuntungan karena setter memberi kita kendali penuh dalam menetapkan nilai ke atribut dan membantu kita membatasi perubahan yang tidak perlu.

```
// Implementing the bird class
public class Bird {

    // Few properties which
    // define the bird
    String animal = "All Birds";
    String color;
    int legs;

    // Implementing the getters and
    // setters for the color and legs.

    public void setColor(String color)
    {
        this.color = color;
    }

    public String getColor()
    {
        return this.color;
    }

    public void setLegs(int legs)
    {
        this.legs = legs;
    }

    public int getLegs()
    {
        return this.legs;
    }

    // Few operations which the
    // bird performs
    public void eat()
    {
        System.out.println(
            "This bird has eaten");
    }

    public void fly()
    {
        System.out.println(
            "This bird is flying");
    }
}
```

- **POLYMORPHISM**

Sekarang, kita ingin membuat merpati dapat terbang dengan menggunakan method “fly” namun kita ingin mendefinisikan tujuan terbang merpati tersebut dan kita ingin mendefinisikan makanan yang dimakan oleh merpati tersebut. Kita dapat melakukannya dengan menggunakan konsep overload method.

```
// Creating the Pigeon class which
// extends the bird class
public class Pigeon extends Bird {

    String animal ="Pigeon";
    // Overriding the fly method
    // which makes this pigeon fly
    @Override
    public void fly() //override method
    {
        System.out.println(
            "Pigeon flys!!!");
    }

    //overload method to method in this class
    public void fly(String place){
        System.out.println("Pegeon flys to "+ place);
    }

    //overload method to method in parent class
    public void eat(String food)
    {
        System.out.println(
            "Pigeon eats "+food);
    }

    //super & this
    public void about(){
        System.out.println(super.animal+" live freely in the sky");
        System.out.println(this.animal+ " can recognise each letter of the human alphabet");
    }
}
```

- **TESTING**

Setelah kita selesai membuat kelas “Bird” dan kelas “Pigeon”, kita perlu membuat sebuat kelas untuk menjalankan atau memanggil kelas-kelas tersebut beserta method-methodnya.

```
//encapsulation, inheritance, & polymorphism testing
public class OOPTest {
    public static void main(String args[]){
        Bird bird = new Bird();
        Pigeon pigeon = new Pigeon();

        bird.eat();
        bird.fly();

        System.out.println("");

        //polymorphism
        pigeon.eat("Seed");
        pigeon.fly();
        pigeon.fly("Sky");

        System.out.println("");

        //Mutators & inheritance
        pigeon.setColor("grey");
        pigeon.setLegs(2);
        System.out.println(
                "this pigeon has "+pigeon.getColor()+" color "
                + "and "+ pigeon.getLegs()+" legs");
        pigeon.eat();

        System.out.println("");
        pigeon.about();

    }
}
```

- **Tugas**

  Dalam sebuah sistem akademik terdapat entitas Mahasiswa. Semua mahasiswa memiliki properti yang umum yaitu Nama dan NPM serta semua properti ini dapat diisi sesuai data mahasiswa. Semua mahasiswa juga dapat mengerjakan tugas dan mengikuti UTS. Dari entitas mahasiswa terdapat entitas mahasiswa yang mengambil mata kuliah PBO. Bagi mahasiswa yang mengikuti matakuliah PBO, sistem dapat mencetak data mahasiswa yaitu Nama dan NPM serta status bahwa mahasiswa tersebut telah mengikuti UTS dengan matakuliah PBO. Di method yang berbeda, program dapat mencetak nilai UTS serta Tugas mahasiswa terebut.